Spring, 2016 CIT 590

Programming Languages and Techniques

Homework 5

all deadlines on canvas

This homework deals with the following topics

* files - reading and writing

* more unit testing

* spell checking

The assignment this time is to design a spell-checking tool that interactively corrects misspelled words in a document. The user gets to select the correct spelling of a misspelled word from a list of candidates provided by the program.

The spell checking tool will perform three tasks:

1. Spot the misspelled words in a file by checking each word in the file against a provided dictionary (note this is not a Python dictionary)
2. Provide the user with a list of alternative words to replace any misspelled word.
3. Write a new file with the corrected words as selected by the user. Note: It would take some work to maintain the original files punctuation. Dont worry about that for this assignment.

A list of correctly spelled English words is provided as a reference in the file called engDictionary, one word per line of the file. The user file to be checked is a text file where each space-separated word of the file is to be examined. As previously noted, punctuation and case should be ignored.

If a word from the file does not exist in the provided list then it is assumed to be misspelled and a set of alternatives provided to the user. The user has three options for the provided candidate list:

- select, by number, one of the candidates. The word in the file is replaced in the output file by the selected number. For this, the user first types the letter 'r'(for replace) and then is prompted to enter a number.
- indicate that they wish to leave the word as is. For this, ask the user to just type the letter 'a'.

- indicate that they wish to provide the alternative by typing it in directly. The program replaces the word in the file with the user provided replacement in the output file. For this, ask the user to type the word 't'(for I will type).

So typical interaction will be something like the following. The blue text indicates what is actually displayed to the user. The normal (black text) is just for explanation purposes

The word 'morbit' is misspelled.
The following suggestions are available
1. 'morbid'
2. 'orbit'.
Press 'r' for replace, 'a' for accept as is, 't' for type in manually.
User presses 'r'
Your word will now be replaced with one of the suggestions
Enter the number corresponding to the word that you want to use for replacement.
User enters 1
In the output file, the morbit will be replaced with morbid.

The word 'automagically' is misspelled.
The following suggestions are available
1. automatically.
Press 'r' for replace, 'a' for accept as is, 't' for type in manually.
User presses 'a'
In the output file, automagically stays automagically

The word 'ewook' is misspelled. The following suggestions are available
1. wok
2. woo
3. awoke
Press 'r' for replace, 'a' for accept as is, 't' for type in manually.
User presses 't'
Please type the word that will be used as the replacement in the output file
User types 'ewok'
In the output file, this word is changed to ewok.

The rare case when the spell checker cannot come up with anything at all. Note that this case is slightly different so be aware of it when you program.

The word 'sleepyyyyyyyyy' is misspelled.
There are 0 suggestions in our dictionary for this word.
Press 'a' for accept as is, 't' for type in manually.
User presses 't'
Please type the word that will be used as the replacement in the output file
User types 'sleepy'

In the output file, this word is changed to sleepy.

Also if the user has a typo or is trying to be silly, and enter something other than 'r', 'a' or 't' and/or a number that is outside of the range of options when they have decided to do a replace, please tell the user politely that they have to try again. Do not let the program exit or crash in a user unfriendly manner.

Also if the user decides to type in a word and then types something, just accept that. If the user is deciding to go ahead and ruin their file with some garbage input, it is not your job to help them out.

The word 'immmmediate' is misspelled.
Press 'a' for accept as is, 't' for type in manually.
User presses 't'
Please type the word that will be used as the replacement in the output file
User types 'T1000R2D2'
In the output file, this word is changed to T1000R2D2.

In the next section we describe all the functions we would like you to implement.

You need to set some rules to select the best word candidates. We provide some basic ideas but for the sake of seeing some creativity, we allow you to come up with your version that uses the results of the helper functions as well. Read on for what these basic ideas are.

You assignment will do the following:

1. Prompt the user for the name of the file to spell check. The program will spell check each word and then write a new file with the name of the original file plus the text -chk. Thus if the file being checked is file.txt, the spell checked output will be file-chk.txt. Note that the file suffix is preserved!

2. Prompt the user for the name of the reference list file. It should default to the provided file engDictionary.txt. This file consists of simply words, one word per line and you can assume any alternate file will have the same format.

3. Create a function that checks a word of the user file against words from the reference file and see if that word is there (more details below in the functions list)

4. Write a loop which reads from the input file and goes through it a word at a time. Each word of the file is checked to see if it is spelled correctly based on the reference list. If it is spelled correctly, just write it directly to the output file. If it is not spelled correctly, then provide the user with the options discussed above. Depending upon their responses, write the corresponding word to the output file.

5. Output the name of the updated spell-checked file. The user can then open up that file via Finder/Windows Explorer (your python program does not need to do this) and see the result of the spell checking.

useful python functions

- the lower() function is important when you read each word, because no match case is required in this assignment.
- As in the previous assignment you will have to use lists, dictionaries, tuples and maybe even sets depending upon the function.
- Remember the split() and the strip() functions. They are good to get rid of unnecessary characters attached to a word, such as , /)(. and all other punctuations.
- Any of the functions from lists, strings, tuples, sets, dictionaries are fair game for this HW. Look through the documentation on python.org. Look through the textbook. As HW4 hopefully demonstrated, there are many ways of writing code in Python. If you know the right functions, it is fewer lines of code.

Functions to implement

Here is the list of functions that you HAVE to implement. For each function you write that does not have input or output, you are expected to have unit tests. Remember that the goal is to have testable code. If you can write a function without having input or output, that design is more favourable than one that does have input/output. Please come to office hours and/or post on piazza if you feel there is no way that you can unit test a function. Usually it will mean that you are doing some extra work in a function and we can help you think through your design.

Also, regardless of the function you have to provide **docStrings**. Again, a docString is a triply quoted string that is right after the function definition. If you do not provide this, we will deduct points in this assignment.

1. **ignoreCaseAndPunc(word)** - given a word that might have upper and lower case letters and punctuation, return a word that is entirely in lower case and all punctuation removed.

   ignoreCaseAndPunc('Actually,') return 'actually'

   ignoreCaseAndPunc('variables.') return 'variables'

   Does this mean the final document will be gramatically incorrect? Yes!

   Do we (me and the TAs) care? No!

   The punctuation that you need to worry about are comma(,), semi-colon(;), colon(:), question mark(?), exclamation (!) and full stop (.)

   Any punctuation outside of this is going to be tolerated. In particular if you have a hyphenated word, just let that go through. So a word like co-exits will probably not get any suggestions because there are no hyphenated words in the dictionary. So even though you probably want to correct it to be co-exist, you will probably just have to type that in yourself.

2. **findWordInDictionary(word, fileName)** - check and see if the word is present in the fileName. You should call ignoreCaseAndPunc on the word before trying to find it in the list. Both word and fileName are going to be strings.

   This function just returns the boolean value True or False.

3. **getWordsOfSimLength(word, fileName, n)** - given a word, return a list of words from the fileName that all have the length +/- a value n.

   For instance if the file 'oed.txt' contains 'ban', 'bang', 'gang', 'aa', 'mange' (on separate lines) then the call to getWordsOfSimLength('ging', 'oed.txt' , 1) should return ['ban', 'bang', 'gang', 'mange']

4. **getWordsWithSameStart(word, wordList, n)** - given a word and a list of words, return a list of words that have at least the first n characters the same.

   To continue with the example above

   getWordsWithSameStart('ging', wordList, 1) will return ['gang']

   getWordsWithSameStart('band', wordList, 2) will return ['ban', 'bang']

5. **getWordsWithCommonLetters(word, wordList, n)** - given a word, return a list of words that have n or more letters in common. Consider only the distinct letters.

   Using the same wordList of ['ban', 'bang', 'gang', 'aa', 'mange']

   getWordsWithCommonLetters('clang', wordList, 3) will return ['gang']

   getWordsWithCommonLetters('immediate', wordList, 3) will return ['mange']

6. **getSimilarityMetric(word1, word2)** - given two words, this function computes two measures of similarity and returns the average.

   leftSimilarity (term I made up) - the number of letters that match up between word1 and word2 as we go from left to right.

   So the leftSimilarity for 'oblige' and 'oblivion' is 4

   the leftSimilarity for 'aghast' and 'gross' is 1

   rightSimilarity(again term I made up) - the number of letters that match up, but this time going from right to left.

   So the rightSimilarity for 'oblige' and 'oblivion' is 1

   the rightSimilarity for 'aghast' and 'gross' is 2

   Then take the average of leftSimilarity and rightSimilarity and return that value

   So getSimilarityMetric('oblige', 'oblivion') will return (4+1)/2.0 = 2.5

7. **getSimilarityDict(word, wordList)** - given a single word, go through every word in the list of words provided by wordList and create what we will call a similarityDictionary. That dictionary has the individual words in wordList as keys and the result of getSimilarityMetric as the values.

   Using the same wordList of ['ban', 'bang', 'gang', 'aa', 'mange']

   then getSimilarityDict('band', wordList) returns the following dictionary

   {'ban': 1.5, 'bang': 3, 'gang': 2, 'aa': 0.5, 'mange': 1.5}

8. **getBestWords(similarityDictionary, n)**

   This function takes a similarityDictionary (see above for definition) and it returns the top n in terms of the similarity.

   Since sorting a dictionary is not easy, here is how you do it

   ```
   listOfTuples = dict1.items()
   listOfTuples.sort(sortIn2D, reverse=True)
   return getListOfFirstComponents(listOfTuples)[0:n]
   ```

   Now you are probably wondering what sortIn2D is? That is another function that you have to write in order to do this custom sorting. We actually saw an example of this in class. Go through the lists subfolder on dropbox and look for fancySort.py to see how you can do this.

   So you do have to write another function called **sortIn2D(tup1, tup2)** which takes in two tuples both of which have two components and it does the comparison based on just the second component.

   Also you will notice another helper function getListOfFirstComponents

   getListOfFirstComponents(tupleList) takes in a list of tuples and returns another list which has just the first components of the tuples.

   getListOfFirstComponents([(1,2), (3,4)]) will return [1,3]

9. **getWordSuggestionsV1(word, fileName, n, commonPercent, topN)** - given an incorrect word, return a list of legal word suggestions as per an (somewhat simple) algorithm given below. Safely assume this function will only be called with a word that is not present in the dictionary.

   To come up with a list of candidate words we first come up with a list of candidate words that satisfy both of these two criteria

   a) +/- n in length. b) have at least commonPercent% of the letters in common

   then for the words that satisfy these two criteria order them based on the similarity metric and return the topN number of them.

   return a list of these topN suggestions

10. **getWordSuggestionsV2(word, fileName, n, topN)** - given an incorrect word, return a list of word suggestions that can all be found the file provided by fileName based on the algorithm

    find words that are within +/- 1 in length with respect to the given word. Note that in this case we are insisting that the suggested words are within a certain length range.

    find words that begin with the same n letters as the given word.

    find words that end with the same n letters as the given word.

    make a list that contains the words that are in **all** these lists

    now order the list based on the word similarity measure that we defined previously and return a list of the topN words based on that.

    getWordSuggestionsV2('biger', 'engDictionary.txt', 2, 2) should return something like ['bigger', 'biker'] for instance.

11. **getCombinedWordSuggestions(word, fileName)** - combine the list of suggestions provided by the two functions above in the following manner

    Take 7 suggestions from each function. Use 75% as the threshold for the first algorithm

    ```
    lst1 = getWordSuggestionsV1(word, fileName, 2, 75, 7)

    lst2 = getWordSuggestionsV2(word, fileName, 1, 7)
    ```

    then do the following steps

    a) combine the two lists and remove duplicates

    b) rank this new list using getSimilarityDict and getBestWords

    c) return the top 10 as a list. If there are fewer than 10 suggestions return all.

    getCombinedWordSuggestions('paul', 'engDictionary.txt') will return something like ['pale', 'hall', 'pail', 'alum', 'plate', 'pull', 'petal', 'peel', 'prowl']

    If you get less than 10 suggestions by combining these two lists, that is fine just provide the user with the suggestions that you get.

12. **prettyPrint(lst)** this is a purely output function. Just to help the user identify the suggestions with a number.

    prettyPrint(['biker', 'tiger', 'bigger']) will print out

    1. biker

    2. tiger

    3. bigger

    this is just a convenience function.

    this function does not return anything and does not need to be unit tested.

O frabjuous day!

Finally, for fun, we want you to take a file which has some spelling mistakes (all intentional) and replace each incorrect word with the 'top' word in your spell checking as per the top word in getCombinedWordSuggestions().

Since this is something that is reusable, write a function to do this. Name the function something reasonable and provide it with some reasonable arguments (by the way in the next assignment we will let you have a lot more freedom in design)

The file provided is called 'jabberwocky'. Some of you might have seen it before (Lewis Carol is awesome!). Since it represents a classic piece of nonsense poetry, it comes with a ton of 'spelling mistakes'.

In case there are no suggestions provided at all, just retain the word as is. For instance, I am not sure there will be a good suggestions for Jabberwocky.

What to submit?

You have to submit 3 things in this assignment. Canvas allows for multiple file uploads, so just submit these 3 things.

spellChecker.py
spellCheckerTests.py
correctJabberwocky.txt (the spell checked file that uses the top word for each 'mistake')

Evaluation

This sums up to 30 points. You will get a score out of 30 that then will be scaled down to 20 points. The overall weight of this assignment is the same as the others.

- Good user interface - Can your program actually be used for spell checking 5 pts
- Following specifications and getting all the functions to work 15 pts
- Documentation - all your functions should have docStrings and they should clearly describe what the function is doing. This is true of any helper functions that you write as well. This is worth 5 pts.
- Unit tests 10 pts. Please note that for your unit test you are best off if you write a setUp function that just takes some made up data.